MATH 557 Oct 22

Computable functions

Turing machines

A **Turing machine** M has an unlimited (both sides) **tape** which is partitioned into infinitely many **cells**. Each cell can hold a letter from a finite **alphabet** $s_1, \dots, s_n$. At any time, only finitely many cells contain a symbol $\in \{s_1, \dots, s_n\}$. The empty cells are marked by a *blank* symbol b that is distinct from the s_i. To facilitate things, we put $s_0 := b$.

Access to the tape is given via a **read-write head**, which scans a single cell. The cell currently scanned can be read and its contents overwritten. The head can also move left or right. These actions can be made dependent on the contents of the scanned cell.

Thus, the machine can read any cell on the tape and read and write its contents. Furthermore, at any time M is in one of finitely many **states** $q_1, \dots, q_r$. The states influence the behavior of M.

Formally, there are three types of operations (also called *instructions*): Suppose M is in state q_i and currently scan a cell with symbol s_j $(0 \leq j \leq n)$.

1. $q_i\ s_j\ s_k\ q_l$: delete s_j write s_k,
2. $q_i\ s_j\ R\ q_l$: move the head one cell to the right,
3. $q_i\ s_j\ L\ q_l$: move the head one cell to the left.

In each case, transition to q_l afterwards.

A (deterministic) **Turing program** is a finite list of instructions (1)-(3) such that for any pair (q_i, s_k) there is at most one instruction starting with q_i, s_k.

The computation of a Turing program

The machine begins in state q_1 scanning the leftmost cell not containing a blank and follows the instructions, describe above, according to the current state and the content of the current cell.

If the machine enters state q_r, the computation **halts** (no further instructions will be applied). If it ever enters a state q_i and scans symbol s_j for which there is no applicable instruction, it **stalls** (in particular, there will be no output as defined below).

During the computation, the machine will pass from one **configuration** to the next. A configuration consists of

> the current state + the current cell scanned + the contents the left and right of the current cell up to the leftmost/rightmost non-blank symbol.

Every configuration is determined by a finite amount of information.

An example

The alphabet of M consists of the symbols $0, 1$ (and b for a blank cell), the states are q_1, q_2, q_3. The program P has the following instructions:

$$q_1\ 0\ R\ q_1$$
$$q_1\ 1\ 0\ q_2$$
$$q_2\ 0\ R\ q_2$$
$$q_2\ 1\ R\ q_1$$
$$q_2\ b\ b\ q_3$$

In the initial configuration, the machine looks as follows:

				↓									
…	b	b	b	1	1	1	1	1	1	b	b	b	…

M follows the instructions above until it reaches the halting state q_3

										↓			
…	b	b	b	0	1	0	1	0	1	b	b	b	…

Turing-computable functions

Since Turing computations may not halt (either because they stall or enter an infinite loop and never reach the halting state), we define **partial computable functions** whose domain may be a proper subset of $\mathbb{N}^n$.

We use the following notation:

$$f(\vec{x})\downarrow :\iff f \text{ is defined for } \vec{x}$$
$$f(\vec{x})\uparrow :\iff f \text{ is not defined for } \vec{x}$$
$$f(\vec{x}) \simeq g(\vec{x}) :\iff f(\vec{x})\downarrow \iff g(\vec{x})\downarrow \quad \text{and} \quad f(\vec{x})\downarrow \implies f(\vec{x}) = g(\vec{x}).$$

Definition 0.1.
We say a Turing machine M with program P **computes** a partial function $f :\subseteq \mathbb{N} \to \mathbb{N}$ if the machine, when starting with $n+1$ many 1's, ends in state q_r if and only if n is in the domain of f, and in this case, when M reaches state q_r, there are exactly $f(n)$ many 1's on the tape (not necessarily contiguous).

A partial function f is **Turing computable** if and only if there exists a Turing machine M and a program P that computes it.

The definition for multidimensional functions is similar.

Register machines

Sheperdson and *Sturgis* proposed a variant of a machine model in 1963: An **unlimited register machine** (URM) has an infinite number of *registers* $R_1, \ldots, R_l, \ldots$, each containing a natural number; we use r_n to denote the number in register R_n. These numbers are modified by a URM according to the following types of **instructions**:

Instruction	Operation	Alternative Form	Description
$O(n)$:	$0 \to R_n$	or $r_n := 0$	replace r_n with 0
$S(n)$:	$r_n + 1 \to R_n$	or $r_n := r_n + 1$	add 1 to r_n
$T(m,n)$:	$r_m \to R_n$	or $r_n := r_m$	replace r_n with r_m
$J(m,n,q)$:	if $r_n = r_m$, then jump to the qth instruction, otherwise proceed to the next.		

Note: The jump instruction $J(m, n, q)$ spans multiple columns in the original layout since it has a different structure than the other instructions. A **program** is a numbered finite sequence of such instructions, $P = (I_1, \dots, I_k)$. To run a a program, the URM must be provided with an **initial configuration** of the registers, i.e. a sequence $(a_i)_{n\in\mathbb{N}}$ such that $r_i = a_i$ for all i. The machine then executes its program line-by-line, unless it encounters a jump instruction (in which case it performs the test and, if applicable, jumps to the given instruction). The computation **halts** if there is no "next" instruction, that is, either there is no next line to execute or the instruction given in a J-type line does not exist.

Example

$$\begin{aligned} I_1 &: \quad J(1,2,6) \\ I_2 &: \quad\quad S(2) \\ I_3 &: \quad\quad S(3) \\ I_4 &: \quad J(1,2,6) \\ I_5 &: \quad J(1,1,2) \\ I_6 &: \quad\quad T(3,1) \end{aligned}$$

If the URM starts with the number 9 in the 1st and 5 in the 2nd register (and 0 in all others), it produces the number 4 in the 3rd register after several steps and then reaches instruction I_6, which transfers the number 4 to the 1st register; since no further instruction follows, the program halts.

URM-computable functions

Definition 0.2. Let f be a partial n-ary function on the natural numbers, P a program, and let $a_1, \dots, a_n, b$ be given numbers. We say P on input $a_1, \dots, a_n$ **converges to** b, written $P(a_1, \dots, a_n) \downarrow b$, if the run of P with initial configuration $a_1, \dots, a_n, 0, 0, 0, \dots$ halts after finitely many steps and the number b is in the 1st register.

P **computes** f if and only if for all $a_1, \dots, a_n, b$ we have

$$P(a_1, \dots, a_n) \downarrow b \iff f(a_1, \dots, a_n) \text{ is defined and equals } b.$$

We say f is **URM-computable** if and only if there exists a program P that computes f.

Church-Turing Thesis

There are further approaches to formalize the notion of **computable functions**:

- *Gödel-Herbrand-Kleene* (1936): *general recursive functions* defined via an equational calculus
- *Church* (1936): λ-definable functions
- *Gödel-Kleene* (1936): μ-recursive functions
- *Post* (1943): computable functions via *canonical deduction systems*
- Markov (1951): computable functions via *algorithms* over finite alphabets

All these definitions turned out to be formally equivalent, and moreover, every individual (intuitively) computable function has so far been shown to be computable in the above formal sense, which has given rise to the following thesis:

> Church-Turing Thesis
>
> The (intuitively) computable functions are precisely the (in one of the above senses) formally computable functions.

Enumerability of programs and computable functions

We include, without detailed proofs, some basic facts and consequences regarding the enumeration of computable functions.

Since all programs can be effectively enumerated (in any formalization, they are finite sequences over a countable alhpabet), there is an *effective enumeration* $\varphi_0, \varphi_1, \ldots, \varphi_n, \ldots$ of all unary computable (partial) functions. In particular, given an index n, we can effectively produce a (Turing-, URM-, …) program that computes φ_n.

Using a diagonal argument, one obtains from this:

- there exists a total function $f : \mathbb{N} \to \mathbb{N}$ that is *not* computable:

$$f(n) = \begin{cases} \varphi_n(n) + 1 & \text{if } \varphi_n(n) \text{ is defined,} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, while there exists an enumeration of all *total* computable functions, this enumeration itself *cannot* be computable.

The following predicates are also *not decidable* (i.e., their characteristic functions are not computable):

- $\varphi_n(n)$ is defined,
- φ_n is a total function,
- φ_n is the constant function 0,
- $\varphi_n = \varphi_m$,
- the n-th program with input n (more generally with input m) halts after finitely many steps (the **halting problem**).

Universal functions (or programs): For every computable 2-ary function f, there exists a total computable function k with $f(n, m) = \varphi_{k(n)}(m)$; the 2-ary function $\varphi(n, m) = \varphi_n(m)$ is computable.